基于负载预测的微服务混合自动扩展 *

宋程豪，江凌云†

(南京邮电大学 通信与信息工程学院，南京 210003)

摘　要：由于边缘云没有比中心云更强大的计算处理能力，在应对动态负载时很容易导致无意义的扩展抖动或资源处理能力不足的问题，所以本文在一个真实的边缘云环境中对微服务应用程序使用两个合成和两个实际工作负载进行实验评估，并提出了一种基于负载预测的混合自动扩展方法(Predictively Horizontal and Vertical Pod Autoscaling，Pre-HVPA)。该方法首先采用机器学习对负载数据特征进行预测，并获得最终负载预测结果。然后利用预测负载进行水平和垂直的混合自动扩展。仿真结果表明，基于该方法所进行自动扩展可以减少扩展抖动和容器使用数量，所以适用于边缘云环境中的微服务应用。

关键词：边缘云；微服务；负载预测；混合自动扩展

中图分类号：TN929.5　　doi: 10.19734/j.issn.1001-3695.2022.01.0020

Hybrid autoscaling of microservices based on workload prediction

Song Chenghao, Jiang Lingyun†

(*College of Telecommunications & Information Engineering*, *Nanjing University of Posts & Telecommunication*, *Nanjing* 210003, *China*)

Abstract: Since edge clouds are not more powerful than the central cloud, it is easy to lead to unexpected autoscaling or low resource processing capabilities in response to dynamic workload. Therefore, this paper used the microservice application in a real edge cloud environment to experimentally evaluate two synthesis and two actual workloads. And this paper proposed a hybrid autoscaling method based on workload prediction (Predictively Horizontal and Vertical Pod Auto-scaling, Pre-HVPA) , which first uses machine learning to carry out the workload data characteristics. After obtaining the final workload prediction result, this paper use the predictive workload for hybrid autoscaling module. The simulation shows that the microservice autoscaling policy based on this method can reduce more scaled jitter and more pod container use, and the method is scalable, so it is suitable for the microservice applications in edge cloud environment.

Key words: edge computing; microservices; workload prediction; hybrid autoscaling

0　引言

在过去几年里，云计算领域逐渐与物联网相结合，使得边缘云计算(Multi-access Edge Computing，MEC)和采用微服务架构(microservice)的云端自动扩展技术逐渐成为关注的方向。边缘云计算提供了离用户和应用程序更近的微数据中心(Micro Data Center， MDC)设施，这有助于克服应用程序延迟问题。微服务[1]是一种新兴的体系架构，它可以将应用程序构建为多个细粒度的分布式组件的组合，这些组件可以互相交互，以服务于复杂的业务实例。如今，许多工业应用程序，尤其是边缘云和基于物联网的服务，都采用微服务体系架构。边缘云平台采用微服务架构，在带来很多优势的同时也存在着很多问题，例如其处理资源请求的能力就远不如采用微服务架构的中心云。而对于处理资源请求的问题多采用微服务自动扩展进行处理，所以基于边缘云的微服务系统所面临的一个主要挑战是如何在动态变化负载的请求之下，采取有效的方法进行应用程序的自动扩展，从而来降低边缘云资源的消耗[2]。

现阶段，已经有许多工具可以进行微服务的自动扩展，例如当前非常流行的容器编排管理工具 Kubernetes(下文均以简称 k8s 替代)，它允许通过水平自动扩展(Horizontal Pod Scaling，HPA)或者垂直自动扩展(Vertical Pod Scaling，VPA)来适应应用程序部署。其中水平自动扩展允许增加和减少应用程序实例的数量，例如 Pod(k8s 中最小的计量单位)的数量。垂直自动扩展允许增加和减少分配给每个应用程序实例的计算资源。而大多数现有的解决方案仅仅考虑了水平自动扩展[3]或者垂直自动扩展[4]中的一种。但无论采用哪种方式，其扩展所需的资源都非常多。尤其是水平自动扩展，这种扩展技术是将微服务复制到其他机器上以实现高可用性。然而单纯地通过将一个微服务从一台机器复制到另一台机器上，并不是一个节省资源的方法，因为它的资源会在扩展时被复制。所以这种方法是贪婪的，只适用于不缺乏硬件资源的条件下[5]，这显然不适用于粒度细、资源少的边缘云平台。

此外，在容器级别的自动扩展中，k8s 的自动扩展策略是基于阈值的[6]。这种方法在应对动态负载时会产生较多的伸缩抖动，从而产生额外的资源开销[7]。尽管目前有一些研究对 k8s 的自动扩展策略进行了改进研究，采用了基于负载预测的方法，其性能相较于基于阈值的自动扩展策略获得了一定的提升，但他们的方法对于处理能力较低的边缘云平台并不适用。此外较低的负载波动也不能很好体现扩展的效果。

所以，针对上述问题，如何找到一种在边缘云平台低资源消耗，细粒度的自动扩展策略是目前研究的重难点之一。

1　相关工作

目前，国内外学者已经提出了一些关于微服务自动扩展策略与算法。

收稿日期：2022-01-05；**修回日期：**2022-03-10　**基金项目：**江苏省重点研发项目(BE2020084-4)

作者简介：宋程豪(1997-)，男，山东淄博人，硕士研究生，主要研究方向为云计算和微服务自动扩展；江凌云(1971-)，女(通信作者)，安徽安庆人，副教授，硕导，硕士，主要研究方向为下一代网络(jiangly@njupt.edu.cn).

文献[8]，[10]皆采用基于阈值的策略，这是自动扩展最流行的方法，在 k8s 框架中也是如此。但是确定良好的阈值是一项具有挑战性的工作，在容器级别上需要调整诸多的缩放参数才能确定一个最优的阈值来进行自动扩展。此外，k8s 还增加了对垂直自动缩放器的支持，然而，在发布时，它仍然是一个测试性的概念，并没有关于垂直自动扩展实现[16]的具体细节。

针对 k8s 所存在的问题。文献[3]基于 Q 理论提出了一种新颖的算法，用于自适应和动态地调整阈值，并在满足 SLA 目标时不需要用户进行额外配置。该方法在一定程度上改进了现有水平自动扩展关于难以确定阈值的问题，但是对于变化迅速的工作负载效果则并不好。文献[9] 利用人工神经网络和资源优化方法来预测工作量，并用于运行在云中基础设施上的微服务自动扩展。该方法实现了应用程序经济有效的自动扩展。文献[12]提出了一种窗口预测的方法，对雾中的微服务应用进行自动扩展，并改善了响应时间的SLO。文献[13]虽然使用 ARIMA 模型对负载作出短期预测，相比基于阈值的自动扩展作出了改进，但是它的扩展策略仍然采用的是水平自动扩展，而单纯复制 Pod 副本这种粗粒度的水平扩展方法不适用于边缘云平台。上述研究的改进均聚焦于对基于阈值扩展的弊端，并没有考虑扩展的粒度以及维度。

事实上，大多数现有的自动扩展策略仍然是水平或垂直的。文献[11]基于强化学习提出了一种提升 QOS 的算法，用于改善 k8s 水平扩展的响应时间过长问题，并在应用程序满足响应时间时简化用户的配置过程。该方法使用灾难管理系统进行测试与验证，在一定程度上改进了现有水平自动扩展响应时间的问题，但是灾难管理系统带来的工作负载过大，不适合边缘云中的应用程序。文献[15]提出了一个框架，该框架通过监测收集容器的资源利用率来进行自动扩展，并考虑到波动的容器数量。然而他们采用的仍然是单维度的自动扩展方法，并没有考虑更加细粒度的扩展策略。

到目前为止，只有部分文献考虑将两个扩展维度结合到容器的应用程序中。文献[14]在更细粒度的混合自动扩展上取得了成效，并通过强化学习方法优化了响应时间。文献[4]使用 IBM 自主计算的 Mape-K 原理进行垂直扩展，通过充分利用垂直弹性，应用程序可以通过细粒度的垂直扩展更多减少容器的使用数量，同时也可以通过水平扩展应对工作负载高峰。CloudVAMP[17]是一个为 VM 提供内存超额订阅机制的平台。内存超额订阅允许虚拟机使用比主机可用内存更多的内存，即垂直扩展。然而 CloudVAMP 不支持对不同类型的负载进行扩展，从而丧失了实现更细粒度资源管理的能力。上述文献与本文的工作不同，本文重点考虑的是边缘节点的微服务进行自动扩展时的资源消耗问题。

本文主要基于[13]，[14]文献进行改进，对基于预测后的负载结果采取了混合自动扩展的策略，从而减少扩展的容器副本数量(Pod 副本数量)以及扩展抖动的次数。

2　边缘云微服务平台部署

2.1　部署框架

在边缘云环境中，首先将中心云与边缘云平台的模块进行重新配置和依赖分解，从而实现云平台组件的容器化。然后分别在中心云和边缘云节点安装 k8s 工作负载节点。最后在完成基础设施编排服务、k8s 集群和边缘云节点的配置后，再对云端连接组件进行定义，实现中心云数据与边缘云计算节点之间的信息控制。在中心云与边缘云部署 k8s 的实现如图 1。为了更好的体现本文在边缘云的自动扩展策略，下面对当下最流行的解决方案的扩展流程进行简单的阐述并将其在后续作为基准方案进行对比与衡量。

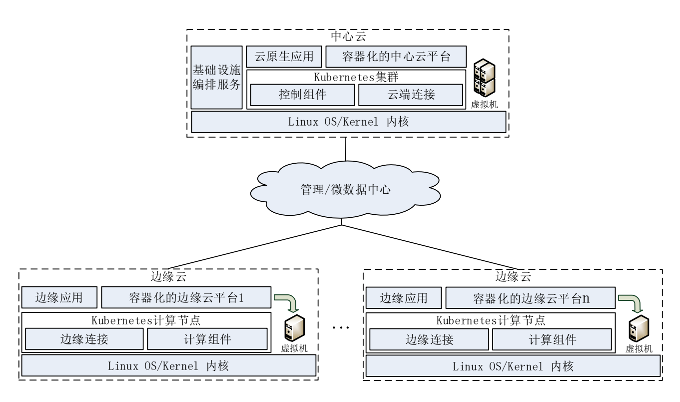


图 1　边缘云部署框架图

Fig. 1　Edge cloud deployment framework

2.2　k8s 自动扩展方案

k8s 作为现在最流行的容器解决方案，提供了服务发现，资源调度，监控，应用部署与自动扩展等服务。它在微服务应用程序自动扩展方面给出的解决方案是基于阈值的水平自动扩展。这种扩展策略存在两个问题：采用阈值进行自动扩展时的问题在于很难确定一个良好的阈值；水平自动扩展的粗粒度，是在给定 Pod 的 CPU 或者内存利用率(所请求相应资源数量的百分比)超过上限阈值或者低于下限时进行扩展和缩减的过程，而这个过程中单个 Pod 的利用率并没有达到最优，即存在多个 Pod 扩展和缩减后的资源没有完全使用。

算法 1　kubernetes 水平自动扩展

1) 输入：应用负载资源请求总数 $Usage_t$， T 扩展阈值，Pod 的资源需求值 $Request_t$；

2) 输出：Pod 扩展的数量 $NumPods$；

3) 计算扩展与缩减阈值 $\tau_r^+ = T + (1 + tolreance)$，$\tau_r^- = T + (1 - tolreance)$，其中 $tolreance$ 为默认值 0.1;

4) 计算资源利用率 $Utilization = \dfrac{Usage_r}{Request_t}$；

5) 判断 $Utilization > \tau_r^+$，若满足该条件则 Pod 数量 $NumPods + 1$，反之则 $NumPods - 1$；

6) 最终统计 $NumPods$ 扩展数量的结果。

k8s 自动扩展算法(见算法 1)利用水平自动扩展来调整 Pod 副本的数量，以满足当前传入的负载请求。该算法(第 4 章命名为 Kubernetes)是根据当前资源利用率增加或减少微服务实例 Pod 的数量。如果一个微服务及其所有副本的平均资源利用率超过某个目标百分比阈值，那系统将扩大 Pod 副本的数量。相反，当平均资源利用率低于阈值时，系统将减少 Pod 副本的数量。资源利用率计算为资源分配量超过请求总数。为了获取平均资源利用率，本文设置每 5s 获取一次资源请求数量，并计算对应时刻的平均资源利用率。该策略为下文改进方案时的基准方案，其大致实现架构见图 2。

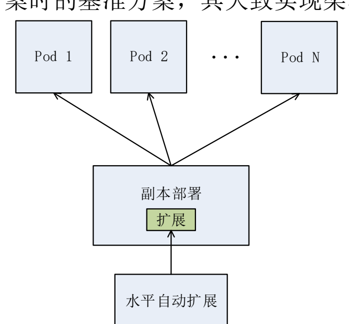


图 2　Kubernetes 水平自动扩展过程

Fig. 2　Kubernetes horizontal pod autoscaling schematic

2.3　预测式与响应式自动扩展算法的局限性

在现有文献中，通常只是提出预测方法改进 k8s 的反映式扩展带来的抖动问题，或将问题的重心放在通过改变扩展

策略来优化其响应时间的违规次数。例如，本文对比预测式算法[13](在第 4 章仿真分析中命名为 HPA)采用了 ARIMA 的短期预测算法，其输入为应用的资源利用率，输出即为 ARIMA 预测自动扩展值，通过预测时间序列模型即对资源利用率进行均值化处理，并利用处理后的结果计算自相关和偏自相关函数得到结果值。这种方法存在的局限性在于它只是对导入微服务的负载采取预测，尽管在抖动的次数上取得了优化，但是提前预知并作出扩展会导致使用更多的 Pods 数量，从而使用更多的计算资源。本文对比响应式混合自动扩展算法[14](在第 4 章命名为 HVPA)采用强化学习进行微服务混合自动扩展，使用基于模型的强化学习更新算法，该算法的本质是从微服务自动扩展的细粒度角度出发，通过 9 步转移模型引入垂直自动扩展。从而可以更精确的扩展微服务。然而引入细粒度扩展却导致了在扩展的过程中采取精确扩展数量为目标，而这样会导致在负载频繁发生改变时，微服务的 Pods 副本也会频繁进行扩展与缩减，从而产生了无谓的伸缩抖动，同样不利于边缘云的资源利用。需要注意的是，k8s 方案为现有的基准方案，对于抖动次数和 Pods 副本扩展数量这两种指标都有待优化，而上述基于负载预测的自动扩展方案以及响应式混合自动扩展方案都只能对其中一种指标进行优化与改进。为了能在两个指标均产生一定的优化，本文利用机器学习对两种真实负载和两种合成负载进行特征预测，并提出了一种混合式自动缩放算法。该算法的优势在于结合了预测算法与混合自动扩展，能提前扩展处理变化迅速的负载，从而减少频繁的扩展抖动；同时通过计算最大最小的副本资源，优先进行垂直扩展，更多利用单个副本的细粒度资源，完成混合式自动扩展，减少了在负载变化相对缓慢时进行扩展所使用的 Pods 数量，最终达到整体减少资源消耗的目的。

3 预测混合自动扩展算法(Pre-HVPA)

为解决自动扩展中 Pod 资源消耗过度和伸缩抖动的问题，本文提出了一种使用机器学习进行负载预测的混合自动扩展方法。图 3 是本文提出的方法框图，其主要步骤如下，首先为部署在边缘设备的微服务生成了增长型和周期型的工作负载并采集了 CPU 密集型负载与 I/O 内存密集型负载的信息。其次通过对负载信息进行划分，使用机器学习对模型进行训练，从而对负载信息进行预测。最后将得到的负载预测结果集成本文所提出的边缘云 k8s 平台的混合自动扩展模块，从而获得所需的 Pod 数量变化以及伸缩次数的统计。

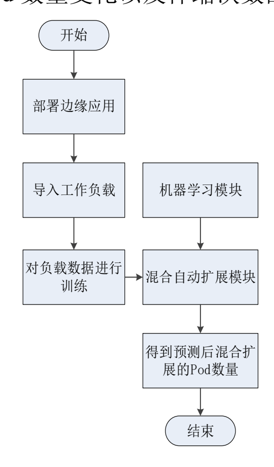


图 3 预测混合扩展方法框图

Fig. 3 Block diagram of predicting hybrid scaling method

3.1 部署边缘应用

本文使用部署在边缘云节点的设备(见4.1 实验环境)来运行微服务应用程序，该应用程序使用基于回归的监督机器学习算法(SVR)，利用从传感器收集的历史日志来预测下一个时间段的温度。这种基准测试的应用程序模拟 CPU 密集型以及内存密集型的工作负载，这是微服务设计的典型业务用例。例如，可以为物联网工业应用中的数据预测、搜索算法、数字音频/视频内容转换、数据压缩任务设计通用的微服务应用。

3.2 导入工作负载

为了评估所使用的预测混合自动扩展模型，本文使用了两种合成负载和两种真实负载。两种合成负载分别为递增型和周期型工作负载，并用于模拟应用程序的常规负载变化情况。当应用程序的大多数用户开始访问服务时，通常都会观察到负载的持续增加。对于周期型负载在实际的应用程序中也是非常常见的，并且会在持续的时间内符合周期性变化。

首先通过初始化固定的请求数量 $M(t_0)$，然后以每分钟固定的增长率 Δq 增加工作负载，具体计算如式(1)所示。

$$M(t)=M(t_0)+(t-1)\Delta q+r_c Rand(\mathbb{N}(0,1)) \tag{1}$$

其中设定 $M(t_0)=100$，$\Delta q=60$，$r_c=30$。式(1)代表在 $M(t)$ 120 分钟内的请求总数。

为生成周期型的工作负载，本文使用了一个周期为 λ，振幅为 α，初始偏置因子为 $U(t_0)$ 的正弦函数。具体计算如式(2)所示。

$$U(t)=U(t_0)+\alpha(\frac{2t\pi}{\lambda})+r_c Rand(\mathbb{N}(0,1)) \tag{2}$$

其中设定 $U(t_0)=3500$，$\alpha=2500$，$\lambda=50$ 来生成周期型工作负载。

关于两种真实负载主要使用了已公开的阿里云天池 2020 混合云的工作负载跟踪来模拟真实世界的工作负载情况。其中包括 CPU 计算密集型负载与 I/O 内存密集型负载。

3.3 对负载数据进行训练、机器学习模块

本文利用 4 种工作负载的数据集，对 5 种不同的线性回归模型进行训练，对比不同的预测模型(包括线性回归模型 LR[18]、弹性网模型 EN[19]、多项式回归模型 PR[20]、XGBoost 模型 XGB[21]、决策树模型 DTR[22])所产生的 MSE(均方误差)。从而评估对本文预测准确性，并最终选取测试数据 MSE 最小的机器学习方法作为本文提出的混合自动扩展模型的学习算法。MSE 的计算方法见式(3)。

$$\mathrm{MSE}=\frac{1}{m}\sum_{i=1}^{m}\left(y_i-y_i\right)^2 \tag{3}$$

该参数可以反映估计值与真实值的差异程度，其中 m 表示样本数量，y_i 表示真实值，$\hat{y}_i$ 表示预测值。通过评估后的 MSE 结果如表 1。

表 1 数据集预测误差 MSE

Tab. 1 Prediction error(MSE) of dataset

Method	MSE	Method	MSE
LR	0.682	DTR	0.212
EN	0.554	XGB	0.126
PR	0.480	**BR**	**0.078**

经过 10 次测试，分别取 7 种方法的 MSE 平均值后，可以看出，BR 机器学习方法对 LR、EN、PR、DTR、XGB 有 88.5%，85.9%，83.8%，63.2%，38.1%的改善。为提升混合自动扩展所得结果的准确性，选取预测负载更好的模型非常重要。所以本文选取 BR 为最终的机器学习算法。

3.4 混合自动扩展算法

将上面训练好的机器学习模型以及对预测后的工作负载数据输入到本文所提出的混合自动缩放模块，进行自动扩展。

本文所提出的混合自动扩展算法主要目标是在微服务之间保持一种高可用性、低消耗性。每个边缘节点以高资源利用率的方式来动态的扩展资源。一些微服务倾向于使用 k8s 进行有效的扩展，然而这会导致更多的资源消耗。所以水平自动扩展并不是总是最佳的解决方案，因为扩展一个新的副

本实例(Pod 容器)会需要更多的资源。此外，在进行混合自动扩展时，仍然需要保持水平自动扩展微服务的状态，因为它需要一个一致性的模型来维护所有副本之间的状态。所以在水平扩展不适用的场景中，最好的扩展策略是能为特定的 Pod 容器带来更多的资源，即垂直自动扩展。本文所提出的混合自动扩展算法与只进行粗粒度水平扩展不同，并不是单纯的进行 Pod 的复制，而是能确定的精确的微服务需求，在保留粗粒度水平自动扩展的同时，加入了所需要的细粒度调整。这将大幅度减少复制 Pod 容器的数量，从而减少边缘云节点的资源浪费。

本文的混合自动扩展对不同的负载类型使用不同的扩展度量标准。对于 CPU 密集型负载本文采取由请求数量计算所得的 CPU 利用率，而内存密集型则以内存利用率为度量，为更好的将不同的负载进行同等量化，后续一致以相应负载的资源请求数量(资源利用率)进行衡量。算法 2 以 CPU 密集型负载为例进行阐述。

算法 2　混合自动扩展算法

输入：预测后的负载资源 $Usage_{pre}$，Pod 需求资源 $Request_t$，Pod 的目标资源利用率 $Target$；

输出：Pod 扩展的数量 $NumPods$、扩展与缩减次数 T；

初始化：$Max_{num}=10$，$Min_{num}=1$，$time=120$；

对观察时间内的资源情况 $Usage_{pre}$ 进行遍历，判断资源当前资源利用率与目标利用率的关系，从而确定现有资源分配是否可进行垂直扩展；

while *true* do

　if $sum(Usage_{pre})==(sum(Request_t)*Target)$　//总体资源分配等于目标利用率

　　扩展策略不更改，进行水平扩展，最大扩展至 Max_{num}

　elif $sum(Usage_{pre})>(sum(Request_t)*Target)$　//总体资源分配大于目标利用率，算法进行资源扩展阶段

$$CPU_{up}==\frac{sum(Usage_{pre})}{Target*0.9}-Request_t,\quad NumPods_{up}$$　//首先将每个 Pod 副本尝试向上垂直扩展，然后执行水平扩展

　else　//总体资源分配小于目标利用率，算法进入资源回收阶段

$$NumPods_{down},\quad CPU_{down}==Request_t-\frac{sum(Usage_{pre})}{Target*0.9}$$　//首先将 Pod 副本水平缩减，然后将每个 Pod 尝试向下垂直扩展，最低将 Pod 数量缩减为 Min_{num}

　end

　for *i in range*(0,*time*)

　　$sumNumPods=sumNumPods+Numpods[i]$

end

统计 Pod 副本的扩展结果

当将预测后的负载导入混合缩放模块后，算法首先判断总体的分配资源与目标利用率的关系，决定是优先执行水平自动扩展或者垂直自动扩展。如果资源大于目标值时可以优先进行垂直自动扩展，提升每个 Pod 的资源使用率，从而在减少水平扩展数量的同时，依然可以满足资源的请求。当资源请求减小时，资源首先进行水平缩减，对 Pod 进行回收，再进行垂直缩减，可以最大程度减少 Pod 的使用数量，达到减少资源的消耗目的。

此外，由于输入的是预测负载，所以在进行自动扩展提供了一定的预测性，即可以在已知未来一段时间资源请求的情况下提前进行自动扩展，避免了许多无意义的扩展，降低了扩展抖动的次数，使得 Pod 副本数量变化更为平滑。

4　仿真

4.1　实验环境

本文使用了一个部署在边缘云中的 Kubernetes 集群来测试所提出的混合自动扩展模型，该集群包括了 4 台物理机器分别搭建了版本互相兼容的 Docker 和 Kubernetes，其详细机器配置以及集群的版本信息如表 2 所示。

表 2　边缘云集群信息

Tab. 2　Equipment information of MEC

计算资源	版本规格	计算资源	版本规格
操作系统	CentorOS7.8	Hard disk	1TB
内存/GB	4	Docker	18.6.3
CPU	i5 4 核	Kubernetes	1.17.4

4.2　负载预测结果分析

为更好的验证预测模型，本文采用十折交叉验证法对所有的工作负载数据进行随机划分，以此生成训练集和测试集，其中 10%的数据作为测试集，90%的数据作为训练集。之所以采用该验证方法是因为通过大量实验表明，十等分数据可以获得最优误差估计。

图 4 显示了实验评估的 4 种不同类型的工作负载在 120 分钟内进行预测后的结果，其中图 4(c)与图 4(d)分别为 CPU 密集型工作负载预测结果和 I/O 内存密集型工作负载预测结果。可以看出，内存密集型负载的请求密度与请求总数均低于 CPU 密集型负载，这是因为内存密集型计算任务相对较少。与两种密集型负载相比图 4(a)增长型与图 4(b)周期型负载显示出了更好的预测结果，这是由于任何的机器学习估计方法都难以非常好的处理学习的突发过程，然而对于这种较难预测的突发密集型负载[23, 24]，本文所采用的预测方法已经产生了相对较好的结果。所以将预测更好的负载导入本文混合自动扩展模块时，可以获得更精确的自动扩展结果。

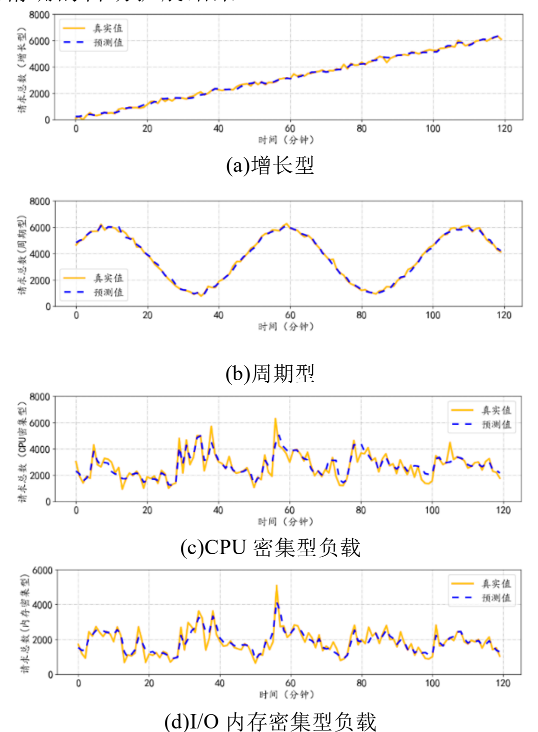


图 4　实际和预测负载实验评估结果

Fig. 4　Actual and forecasted load experiment evaluation

4.3　自动扩展结果分析

在本节中将介绍使用的 k8s 基于阈值水平自动扩展方法，基于负载预测的水平自动扩展方法和基于阈值的反映式混合自动扩展方法。并将本文提出的预测混合自动扩展结果与这些方法在处理递增型、周期型、CPU 密集型负载和内存密集型负载时的自动扩展结果进行对比分析。

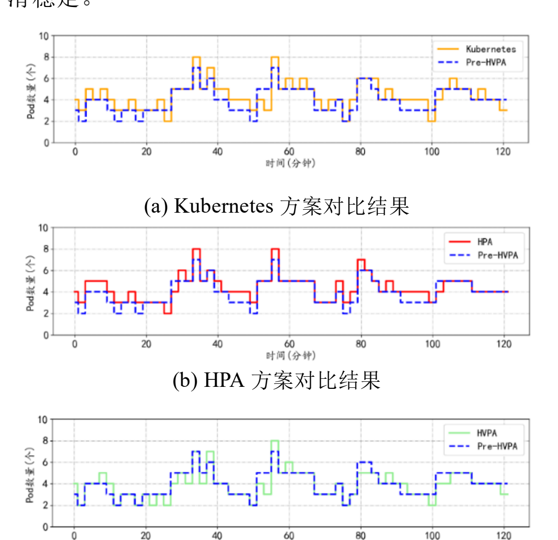
4.3.1 CPU 密集型负载结果分析

K8s 的基础方案是基于阈值的水平自动扩展，该方法属

于反映式的自动扩展策略，当任意 Pod 容器的资源利用率大于用户自定义的扩展阈值(通常 k8s 方案设定为 75%)时，系统会动态向微服务中添加 Pod 容器副本。类似地，当所有容器的资源利用率小于缩减阈值(通常 k8s 方案设定为 50%)时，系统会从已分配的 Pod 容器副本中动态将其移除。通过添加额外的容器，k8s 方案也可以适当地恢复应用程序的性能。但是从图 5(a)可以观察到，在面对 CPU 密集型负载这类计算任务大，变换迅速的负载时，这种方案产生了额外的扩展抖动和使用更多 Pod 副本数量的资源开销。所以本文提出的预测混合自动扩展(Pre-HVPA)方法采用负载预测对 CPU 密集型负载进行特征预测，能更好的适应这类动态变化负载，并且采用细粒度的混合自动扩展，以更精确的资源预测和资源需求计算，获得了比现有最流行的自动扩展方案更良好的结果。

图 5(b)的 HPA 方案是现有文献对 k8s 方案的改进，该方案针对其反映式扩展的弊端，采用了提取 CPU 密集型负载特征来进行预测的方法，提前作出扩展并改善扩展抖动的问题。但是单纯采用预测负载来进行水平自动扩展，会导致使用更多的 Pod 副本数量。可以看到本文提出的方案可以在提前扩展且减少扩展抖动的同时，使用了尽可能少的 Pod 副本数量。可以从图中看出，在 20 到 50 分钟时间段内以及 70 到 80 分钟的时间段内，可以看出本文在预测负载的基础上引入的混合自动扩展策略使用了比水平自动扩展策略更少的 Pod 副本数量。

图 5(c)是与现有提出的基于阈值的反映式混合扩展方案(HVPA)的对比，这种方案在进行扩展时会以最小化数量为目标，而不考虑扩展的抖动，与本文提出的策略相比，在 0 到 40 分钟这段时间，因为对负载变化的进行提前预测，本文预测的混合自动扩展使用了与反映式混合扩展相近的 Pod 副本数量，减少了 2 次无意义的扩展抖动。此外，在 60 分钟负载急剧提升前，本文策略会提前作出扩展决策，也减少了 2 次无谓扩展，使用尽可能少的 Pod 副本数量来使扩展更为平滑稳定。



(a) Kubernetes 方案对比结果

(b) HPA 方案对比结果

(c) HVPA 方案对比结果

图 5　CPU 密集型负载自动扩展结果

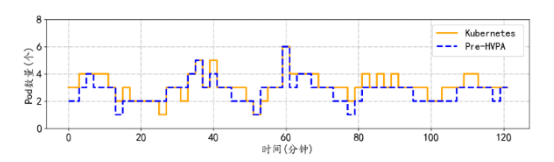
Fig. 5　CPU intensive load autoscaling results

4.3.2 内存密集型负载结果分析

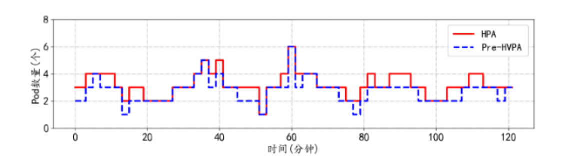
对于更好的印证本文的扩展策略，本文同样衡量了不同类型的负载，即内存密集型负载。这种负载相对于 CPU 密集型负载而言，请求数量相对较少且变化也相对缓慢，同样是有代表意义的一种测试负载类型。

图 6(a)显示本文提出的扩展策略在应对内存密集型负载时得扩展结果，首先可以看到，k8s 方案在应对内存密集型负载时的扩展结果相对于 CPU 密集型变换较为缓慢。即伸缩次数更少，且整体使用的 Pod 副本数量更少。尽管如此，在对这种负载时，本文的预测混合扩展策略仍然体现了较好的优势，在 80 到 100 分钟期间的优势尤为明显，通过提前预测负载变化，减少了 5 次的无意义伸缩抖动，同时也通过更精确的细粒度扩展减少了 Pod 副本数量。对于整体的 Pod 副本数量会在 4.4 节给予更详细的评估。此处不再赘述。

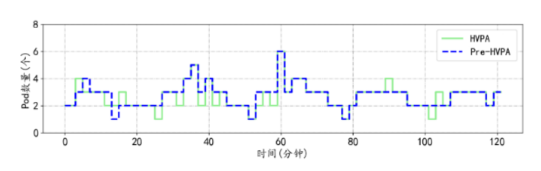
对于 HPA 方案和 HVPA 方案，在图 6(b)于图 6(c)均给出了结果对比，同样在 60 分钟左右负载突然增加时完成了提前的扩展并且相对于其他方案在整体的资源使用上更少，即使用了更少的 Pod 副本数量。



(a) Kubernetes 方案对比结果



(b) HPA 方案对比结果



(c) HVPA 方案对比结果

图 6　内存密集型负载扩展结果

Fig. 6　Memory intensive load autoscaling results

4.3.3 周期型负载结果分析

在应对周期型负载时，由于其变化的规律性是一种比较理想的状态，所以对负载的预测可以更为精确，从而能更准确的计算出资源的请求，提前的进行混合自动扩展策略的部署。实现以更少资源完成扩展的目标。

图 7(a)可以看出相比 k8s 方案时，在 10 到 30 分钟时可以提前进行资源的回收，同时在 40 到 60 分钟进行扩展时，能用相对较多的 Pod 副本数量来进行提前扩展，减少抖动次数。

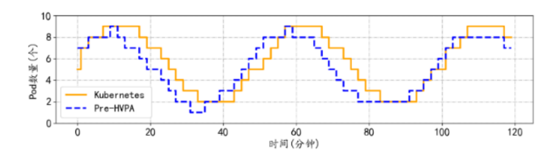
图 7(b)与图 7(c)分别是本文策略与预测式水平自动扩展和混合自动扩展的对比结果，可以看到相比单纯采用预测的扩展方案，本文的策略会使用更少的 Pods 副本数量，而对于单纯采用混合扩展策略的方案扩展的次数更少，即会产生更少的扩展抖动。

此外，还有一种增长型的负载测试结果，与上述不同负载的情况大致相同，此处不再过多分析，其扩展结果将会在 4.4 节对整体系统评估时予以说明。

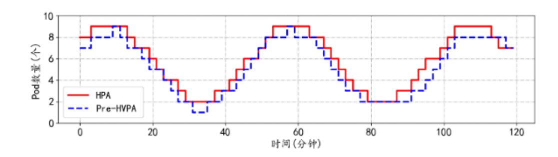
4.4　系统性能评估

本节将对上述 4 种负载的扩展结果从扩展次数与 Pods 使用数量进行评估与分析。图 8 显示了 4 种扩展方案在应对不同类型负载时产生的扩展抖动。可以看出本文的 Pre-HVPA 策略在应对 4 种负载时扩展抖动的次数比 k8s、HPA 和 HVPA 方案少，其中增长型负载对 k8s 和 HVPA 方案分别改善了 40%与 20%；周期型负载对 HVPA 方案改善了 8.5%；CPU 密集型负载对 3 种方案分别改善了 37.2%，18.2%，32.5%；内存密集型负载分别对 3 种方案改善了 18.8%，

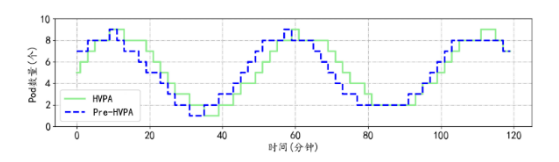
7.1%，25.7%。



(a) Kubernetes 方案对比结果



(b) HPA 方案对比结果



(c) HVPA 方案对比结果

图 7　周期型负载扩展结果

Fig. 7　Periodic load autoscaling results

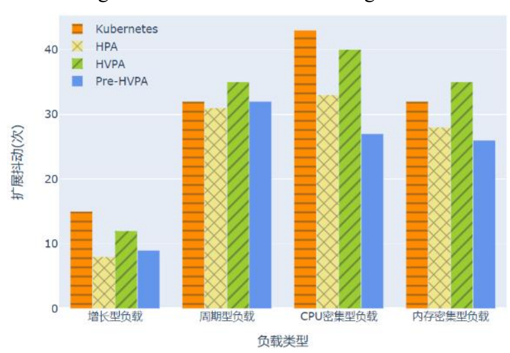


图 8　4 种方案的扩展次数对比

Fig. 8　Comparison of scaled times of four schemes

图 9 是对 120 分钟内扩展 Pod 副本数量进行衡量的结果，可以看出本文提出的策略相较于 k8s、HPA 和 HVPA 方案有明显的改善，其中面对增长型负载对 k8s 和 HPA 方案分别改善了 9.4%，8.4%；周期型负载对于 3 种方案则分别改善了 9.6%，11.4，2.1%；CPU 密集型负载对 k8s 和 HPA 方案分别改善了 9.5%，8.8%；内存密集型负载分别对 k8s 和 HPA 方案改善了 12.4%，14.7%。

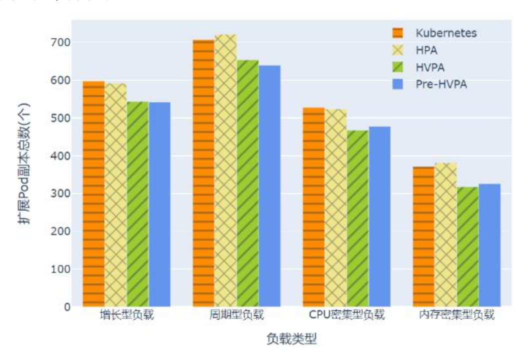


图 9　4 种方案的扩展 Pod 副本数量对比

Fig. 9　Comparison of number of scaled pod replicas of four schemes

在应对增长型负载时，本文的扩展策略相对于 HPA 方案多产生 1 次的扩展抖动，减少了 9.4%Pod 副本使用数量；对于周期型负载，本文策略以与 k8s 方案相同的扩展抖动，减少了 9.6%的 Pod 使用数量；对于 CPU 密集型负载，所提出的方案相对于 HVPA 方案，以牺牲 2.1%的 Pod 数量，减少了 13 次无谓的扩展抖动；而对于内存密集型负载，则以牺牲 2.5%的 Pod 数量减少了 9 次无意义的扩展抖动。

以上是对边缘云环境下的基于负载预测的微服务混合自动扩展策略的仿真结果分析。从系统的综合性能来看，无论是从扩展抖动的性能改善还是 Pod 副本的使用数量的性能改善，所采用的预测式混合自动扩展模型都取得了更好的扩展结果。

5　结束语

现有微服务自动扩展方案大多是部署在中心云的，为边缘云的微服务容器进行资源消耗更少的自动扩展是一项挑战。在本文中提出了一种新的算法，用于为部署在边缘云的微服务工业应用程序确定一种更为经济有效的自动扩展策略。与当前流行的多种方案相比，取得了在扩展次数和扩展 Pod 容器的副本数量上的优化，具有更好的性能。所提出的方法使用 MSE 最小的监督机器学习方法对多种类型的负载进行预测，并将预测后的负载导入本文设计的混合自动扩展模块完成扩展工作。并使用了两种密集型负载和两种合成负载进行实验评估。结果表明，对于两种指标本文提出的策略取得了整体优化。本文的研究内容是一个阶段性成果，在接下来的研究中，本文将尝试使用不同的应用程序来对所提出的模型进行测试，此外对于边缘云的微服务自动扩展而言，如何降低扩展的响应时间，从而减少用户的使用成本也是一个值得研究方向之一。

参考文献：

[1] Taibi D, Lenarduzzi V, Pahl C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation [J]. IEEE Cloud Computing, 2017, 4 (5): 22-32.

[2] Pan J, Mcelhannon J. Future Edge Cloud and Edge Computing for Internet of Things Applications [J]. IEEE Internet of Things Journal, 2018, 5 (1): 439-449.

[3] Horovitz S, Arian Y. Efficient Cloud Auto-Scaling with SLA Objective Using Q-Learning [C]// Proc of IEEE the 6th International Conference on Future Internet of Things and Cloud. Barcelona: IEEE Press, 2018: 85-92.

[4] Dhuraibi Y, Paraiso F, Djarallah N, *et al.* Autonomic vertical elasticity of Docker containers with ELASTICDOCKER [C]// Proc of IEEE the 10th International Conference on Cloud Computing. Honolulu, HI: IEEE Press, 2017: 472-479.

[5] Wilder B. Cloud Architecture Patterns: Using Microsoft Azure [J]. Oreilly & Assoc Inc, 2012.

[6] Lin K, Altaf U, Jayaputera G, *et al.* Auto-Scaling a Defence Application across the Cloud Using Docker and Kubernetes [C]// IEEE/ACM International Conference on Utility and Cloud Computing Companion. Zurich: IEEE Press, 2018: 327-334.

[7] 单朋荣，杨美红，赵志刚，等. 基于 Kubernetes 云平台的弹性伸缩方案设计与实现 [J]. 计算机工程，2021，47 (1)：312-320. (Shan Pengrong, Yang Meihong, Zhao Zhigang, *et al.* Design and Implementation of Elastic Scaling Scheme Based on Kubernetes Cloud Platform [J]. Journal of Computer Engineering, 2021, 47 (1): 312-320.)

[8] Nitto E, Florio L, Tamburri D. Autonomic decentralized microservices: The Gruapproach and its evaluation. [M]// In Microservices: Science and Engineering. Switzerland: Springer Nature, 2020: 209-248

[9] Prachitmutita I, Aittinonmongkol W, Pojjanasuksakul N, *et al.* Auto-scaling microservices on IaaS under SLA with cost-effective framework [C]// Proc of the Tenth International Conference on Advanced Computational Intelligence. Xiamen: IEEE, 2018: 583-588.

[10] Nardelli M, Hochreiner C, Schulte S. Elastic Provisioning of Virtual Machines for Container Deployment [C]// Proc of ACM/SPEC

International Conference on Performance Engineering. New York: ACM/SPEC, 2017: 5-10.

[11] Khaleq A, Ra I. Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications [J]. IEEE Access, 2021, PP (99): 1-13.

[12] Lombardi F, Muti A, Aniello L, *et al.* PASCAL: An architecture for proactive auto-scaling of distributed services [J]. Future Generation Computer Systems, 2019, PP (98): 342-361.

[13] 马小淋. 一种基于负载特征预测的容器云弹性伸缩策略 [J]. 信息安全研究, 2019, 5 (42): 236-241. (Ma Xiaolin. A Container Cloud Elastic Scaling Strategy Based on Load Characteristics Prediction [J]. Journal of Information Security Research, 2019, 5 (42): 236-241.)

[14] Rossi F, Nardelli M, Cardellini V. Horizontal and Vertical Scaling of Container-Based Applications Using Reinforcement Learning [C]// Proc of IEEE the 12th International Conference on Cloud Computing. Milan: IEEE, 2019: 329-338.

[15] Xuxin Tang, Fan Zhang, Xiu Li, *et al.* Quantifying cloud elasticity with container-based autoscaling [J]. Future Generation Computer Systems, 2019, 98: 672–681.

[16] Sembiring K, Beyer A. Dynamic resource allocation for cloud-based media processing [C]// Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video. New York: ACM Press, 2013: 49-54.

[17] Germán M, Miguel C, Carlos A. Automatic memory-based vertical elasticity and oversubscription on cloud platforms [J]. Future Generation Computer Systems, 2016, 56 (1): 1-10.

[18] Olive D J, Linear Regression [M]. Berlin: Springer, 2017.

[19] Zhang Z, Lai Z, Xu Y, Shao L, *et al.* Discriminative Elastic-Net Regularized Linear Regression [J]. IEEE Transactions on Image Processing, 2017, 26 (3): 1466-1481.

[20] Igual L, Seguí S. Introduction to Data Science: Regression Analysis [M]. Switzerland: Springer, 2017: 97-114.

[21] Chen Tianqi, Guestrin C. XGBoost: A Scalable Tree Boosting System [C]// ACM Knowledge Discovery and Data Minin. New York: ACM Press, 2016: 785-794.

[22] Rathore S, Kumar S. A Decision Tree Regression based Approach for the Number of Software Faults Prediction [J]. ACM SIGSOFT Software Engineering Notes, 2016, 41 (1): 1-6.

[23] Ali E A, Seleznjev O, Sjstedt-de S, *et al.* Measuring Cloud Workload Burstiness [C]// IEEE/ACM International Conference on Utility & Cloud Computing. London: IEEE Press, 2014: 566-572.

[24] Ilyushkin A, AHMEDALI E, Herbst N, *et al.* An Experimental Performance Evaluation of Autoscalers for Complex Workflows [J]. ACM Transactions on Modeling and Performance Evaluation of Computing Systems, 2018, 3 (2): 1-32